The Rosebud Concept System

Project Plan
February 6, 1990

A Visionary is merely someone who sees the obvious before the rest of us...

Introduction

This document outlines the project plan for the Rosebud Concept System, or, more simply, Rosebud. Rosebud is the result of investigations and prototyping that goes back to early 1988 and SNARC. SNARC (Sexy New Architecture) was to be a bold new step forward in system software.

This bold step consisted of 2 main risk areas. The first was the creation of a contemporary and powerful new system software foundation. It was recognized that this step would require a significant development effort and risk. This part of SNARC became known as Pink and is progressing well.

The second area was the definition of new application directions. After all, if you were going to take the trouble to create a new software environment, why not also open the door to whole new classes of applications. While Pink has undertaken to create many new application areas, those ideas that were particularly risky became known as Red.

The Red investigation team published a document [1] which suggested that new application areas would be opened with if system software offered significant improvements in 5 key areas:

- Content-aware applications
- Powerful development environment
- User programming and customization
- Enhanced human-human communication
- New user metaphors to enable access to these new capabilities

Family Farm, Ralph and other major ATG projects are tackling many of these issues. The Information Access Concept System (IACS) was created to focus upon the problem of content-aware applications and associated user interface issues. The study group report [2] recommended the construction of a prototype using the concept of Dynamic Books or Binders [3]. This prototype was constructed and demonstrated in the summer and fall of 1989. For further background on the Red project, see [4], [5] and [6].

The IACS became known as Hearst, and the Hearst team made a number of enhancements to the prototype system. The end of 1989 saw a consolidation of efforts among concept system participants. This consolidation saw the Hearst and Institutional Memory concept systems merge. The new group met in late December, 1989, the outline new goals and objectives for the future, In recognition of its new goals and objectives, the team adopted a new name for the concept system: *Rosebud.*

The next section will outline the goals and objectives of the Rosebud team. To meet these goals, the team has designed an architecture to enable content-aware applications. This system architecture will be described in the *System Overview* section. More detail on this architecture will follow in the *Functional Description.* A special aspect of Rosebud is the joint arrangement among Apple, Thinking Machines Corp., Dow Jones and Peat Marwick. The *WAIS* section will discuss this part of the project. The document will conclude with a presentation of the current schedule and task assignments.

Rosebud Goals and Objectives

The Rosebud team met in December 1989 to outline its goals and objectives. This section discusses each of these.

Goals

To design and implement a Concept System which:

- **is a platform for content-aware systems;**

  We believe that applications which are capable of dealing with document contents will be in great demand in the '90s. We want to create a system software framework that will make it easy for developers to create such applications. To that end, we must supply tools and standards.

- **enables the investigation of user interface metaphors for such applications;**

  It is expected that dealing with content-aware applications will result in significant changes to the user interface. New metaphors for dealing with ways to express search criteria and represent results need to be explored.

- **allows us to experiment with the framework(s) to support content-aware applications;**

  The proper framework for content-aware applications is not immediately obvious. We need to create a prototype environment with enough degrees of freedom to experiment with different framework ideas.

- **can be used by ourselves and shared with the "Living Lab";**

  To maximize our experience, we want to share our prototype system with our colleagues. Expansion of the user community will show us how useful the system can be across a broad community. It will also stimulate ideas from outside of the Rosebud team.

- **can become a part of the architecture for future system software;**

  We hope to do more than simply build prototypes. We expect that the Rosebud effort will someday result in production software for future system software products.

- **provides opportunity to obtain feedback from outside Apple through the Apple/TMC relationship;**

  As diverse as ATG and Apple are, we do represent a typical cross-section of the user community. To make our systems as user-centered as possible, we must obtain feedback from those users. The WAIS portion of Rosebud involves a demonstration system in the hands of Peat Marwick users. As current users of information systems, they can provide invaluable feedback as to the utility of our ideas.

  Also, the interactions with the Dow Jones electronic publishing services and the Connection Machine retrieval engines will give us helpful information on cooperating with non-Apple systems.

- **develops open protocols between information clients and servers**

  To truly take advantage of content-aware applications, the user needs to access an information domain well beyond his/her personal files. Huge amounts of electronic information are out there, but currently can't get into people's hands easily. Through the WAIS portion of Rosebud, we will develop protocol standards that will allow any information provider to publish electronic data. Similarly, any information client can retrieve that information. We plan to make these protocols public to help "catalyze the market" [7].

We can also refine these goals in terms of what we will and won't do in the near term.

We will:
- Begin by prototyping on Blue platforms;
- Implement access to a broad range of information sources including Usenet, Dow Jones services, AppleLink and local files;
- Experiment with different user interface models for accessing this information;
- Implement a component framework which allows great freedom in mixing information sources and searching paradigms;
- Develop component protocols to allow this mix-and-match;
- Plan to migrate to Pink as our primary execution vehicle;
- Identify user scenarios and maintain a user centric view.

We won't:
- Plan to develop specific products for Blue;
- Investigate information retrieval technology. Rather, we plan to incorporate existing technologies. We look to Tim Oren's and Steve Weyer's groups for exciting new retrieval technology.

Objectives

We have set both near term and long term objectives. The near term is defined as being within the next 6 months. They are clearly more easily defined and understood. We have also identified objectives beyond the next 6 months. While vague at this point they do point the direction we expect to take the concept system.

In the next 6 months we will:

- **Evolve from work that has already been done**

We do not wish to abandon the effort that was put into the Hearst prototype. We will leverage existing technology where appropriate. For example, we are looking to use the Personal Librarian Software (PLS) product as our immediate source of information retrieval technology.

- **Develop a next generation prototype that we can use ourselves**

  Toward our goal of sharing our technology with the Living Lab, we want to take the current Hearst prototype, known as Phase II and create a Phase III version. This next generation will employ a similar user interface but will be connected to "live" information sources.

- **Spin off a special instance for TMC/Customer to learn from actual user experience**

  The HIG team members will create a portfolio of possible new user interfaces. From that set we will select one most appropriate for the WAIS demonstration system on Peat Marwick premises. That user interface will then be implemented for day to day use on Blue Macintoshes.

- **Explore the user interface issues to uncover the metaphors that work well and perhaps inspire a single "root" paradigm**

  As part of the process that generates the portfolio of user interfaces, the HIG team members are also looking for insight into a new "root" metaphor. That is, a user model which weaves together the different applications' interfaces around a common principle.

Beyond the next 6 months we will:

- **Move the prototype to Pink**

  It remains too early to move the prototype from a Blue platform. Especially since we wish to place this prototype in the hands of non-Apple users at Peat Marwick. However, the next phase of the prototype will be targeted to a Pink platform. As part of that port we expect to discover key areas in which the Pink architecture can help support our framework.

- **Retarget for Ralph with its persistent object model**

  ATG has committed to Ralph as the common ATG development language. We strongly support Ralph and hope to target future prototypes in Ralph. To accomplish that we expect to have a Ralph implementation on Pink within 12 months.

- **Develop a high quality IR engine for inclusion with Pink system software**

  Although information retrieval technology is not of immediate concern, we understand that we would want to incorporate superior IR technology should Rosebud be taken to the product stage. At that point we would want to consider the state of IR research from Tim Oren's and Steve Weyer's groups.

- **Involve third party developers**

  Finally, as a key step toward taking Rosebud to product, we would want to involve key third party developers. For example, Mitch Kapor of On Technology has expressed great interest in having a Rosebud type of framework. His recently announced product, *On Location* currently has to use proprietary content analysis technology. They would rather use built-in tools freeing up more time to spend developing sophisticated user features.

System Overview

This section will offer a brief overview of the architecture of a Rosebud system. It introduces a simple system architecture as shown in the accompanying schematic diagram: *Rosebud Architectural Model.* The terminology is introduced and defined. The following section, *Functional Description*, will build upon these ideas and give the next layer of detail.

The main goal of Rosebud is to produce a framework nurturing content-aware applications. Because of the complexity of issues surrounding content analysis, compression of analyzed text, and the presentation of diverse forms of information, we expect that these applications will in fact be built as a suite of connected components. A particular component may specialize in only one aspect of the problem.

For example, a third party may make use of advanced AI techniques to produce a clever English text "understander". The user could then "plug" it in to a previous system, perhaps by dropping the new analyzer into a "folder" containing the other components. The remainder of the system would operate as before.

In the prototype Rosebud architecture, we have proposed such a framework of interconnected components . Each component is responsible for a relatively small and specialized part of the system . Each component "executes" as if it were its own little virtual computer. Consequently, a component becomes the unit of protection in the event of software faults, and is the unit of processor allocation . The latter refers to the fact that components could execute on different processors of a multiprocessor computer or cooperate across a network of personal computers.

Components would typically be constructed as object-oriented program s. However, they could be implemented in your favorite language since they execute in their own private "virtual computer". Components interact and do not have to be implemented in the same programming language . Rather, they can share information through the exchange of message and the manipulation of shared persistent objects .

The user interface is effected through a component called a "Portrayal". In the theater, a portrayal is a particular interpretation of a character. In Rosebud, the same information may be presented to different users in personalized ways. Similarly, the same user may choose to portray the same information differently depending upon the desired effect. For example, a matrix of numbers may be preferred when doing computations, but a graph may be a preferred portrayal when presenting the same data to others.

The information to be portrayed is collected and organized into components called "Containers" . In the current view, containers are relatively simple entities whose primary purpose is to accept information requests from the portrayal and "map" them into appropriate requests to be forwarded to the actual information sources. The container follows the "instructions" passed along from the portrayal to ensure that only the requested information is passed through into the container.

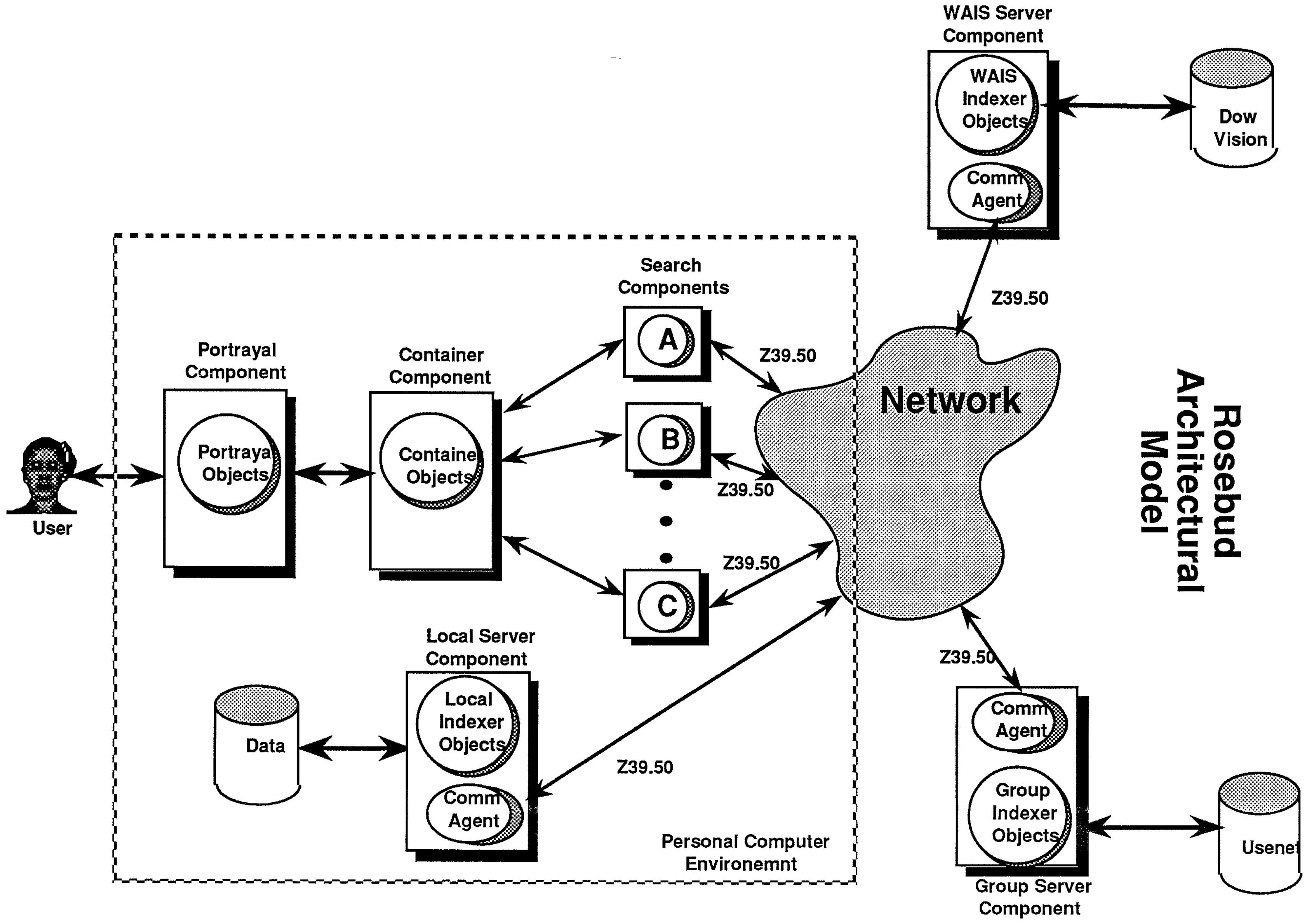
Containers also are responsible for maintaining their contents. For example, if the container happens to hold a reference to an object on a remote computer, and the user wishes to transfer that container to a floppy disk, the container would be responsible for ensuring that that object's contents were copied onto the floppy.

Since the information sources can be quite different from each other, the container access them through "search" components. A search component knows about the particular behavior of an information source and translates the containers search request appropriately. The search component also manages any special communications needed to access a particular source.

The information sources are managed by "server" components. Server components offer a standard protocol to their many clients. Initially, this protocol will be based upon NISO standard Z39.50 [8]. The search components speak the client side of this protocol.

Server components are responsible for collecting raw information from their publishers and processing that information so that it can be retrieved using Z39.50. The actual information may be text or relational data. There is provision for other data types as well. The architecture is symmetric such that a container may also be a server to some other container.

Rosebud Architectural Model



Functional Description

This section will cover the 4 major components, Portrayal, Containers, Search and Server components in more detail.

Portrayal Component

Description

The portrayal is the component responsible for providing the user with a metaphor for interacting with the information. This component is perhaps the most critical to the success of the prototype. Unless the user can comfortably manage the sea of information through an effective user interface, the system will be useless!

The portrayal is responsible for interacting with the user to discover what the user is interested in and what his/her constraints may be. Some of these may be implicit in the portrayal itself. For example, a newspaper portrayal may inherently imply that its corresponding container is to hold information that is no more than 24 hours old.

The portrayal presents to the user the contents of its container. Since one container can collect from other containers, the portrayal can present information from common sources in ways particular to a user's desire. For example, if a user wanted to see the contents of 2 separate containers together, he/she could create a third container which "holds" the contents of the other 2 but presents them in a different manner.

Examples of existing portrayals are the 3-ring binder used by the Hearst Phase II prototype [10], or the dictionary model from the OED prototype [9]. We are unlikely to use either of these portrayals for the next phase. Rather, we are beginning a more intensive investigation of user interface issues. The remainder of this subsection will discuss the process we will be following.

Approach

Our approach to human interface design is based on the creation of user-centered prototypes. These prototypes are informed by study and observation of how people normally perform the tasks which the interface is intended to support. The prototypes are based on users needs, not on the underlying architecture of the software or hardware. Once prototypes have been designed, they are tested with real users. Based on the problems that are uncovered, the prototypes are redesigned and refined. This cyclical process continues through as many iterations as time allows. With each prototype refinement, we get closer to a "shippable" product.

We plan to apply a similar approach to the design of the Rosebud human interface. We will begin by building quick prototypes. The most useful parts of these prototypes will be integrated (using relatively standard Macintosh interface components and metaphors) into a 'vanilla' Macintosh application, which we can put into the hands of end users. This will supply us with information about what sorts of functionality users in the marketplace need. As this information comes in, we will begin a major redesign of the Macintosh interface, with the goal of seamlessly integrating the new information access functionality with the standard Macintosh functionality, and providing a uniform set of interface components and metaphors through which to access it.

Just as each Macintosh application uses the same interface mechanisms for accessing and editing data, it will need uniform mechanisms for accessing, managing, and viewing information. Our ultimate goal is to provide a set of user interface primitives that will serve as the core user interaction elements for information retrieval on the Macintosh. We will also create prototypes which illustrate how these primitives may be combined to achieve that end.

Some Basic Assumptions

The following assumptions underlie the prototypes we will create.

1. The world is diverse. There will be a multitude of databases, and many query mechanisms, available to the average user. Users must be aided in coping with this massive amount of information.

2. Users have an immense variety of constantly changing needs and uses for information. Information that is over five minutes old may be useless to one person, but still valuable to another. The context or history of a piece of information may be essential to one user, but a nuisance or distraction to another. Users will need extremely flexible ways of retrieving, viewing, and managing information.

3. Users will want to access information at any place and time; not just when they're at their desks. For the near future, this implies that the interface must be scaleable: that is, it must work on relatively small screens, with non-mouse input devices.

Our View of Interface Issues

We have uncovered the following set of issues (which we are *sure* are not complete) to address in our prototypes.

What's out there?

How will average users cope with the hundreds or thousands of readily available databases? Somehow users need to be able to browse this multitude of databases, [note that "browsing" is just one possible metaphor], or otherwise be informed of their content.

1. Browsing the data space. How do users figure out which databases to look in?
   Users will be confronted with hundreds or thousands of databases. How can they figure out what subset of the databases are relevant to their particular needs?

2. Browsing a particular database. How does a user know what's in a particular database?
   Users will need ways to get overviews of individual databases which seem of interest. Can a database give a "tour" of itself, or otherwise present its most important features? Can a general set of tools be devised that allow a user to get a reasonable overview? Can these tools also be used to support casual browsing? What types of information are most relevant to users looking for particular databases?

3. Managing database connections. How do users request connection to a specific database?
How is a connection to a database indicated. If this costs users money, how should we allow users to set automatic limits?

Getting and receiving specific information.

Although the ability to browse is essential to an information access system, users won't give a damn about this unless, most of the time, they can get what they want, when they want it.

4. Query building. How can users retrieve specific information?
There needs to be a way—probably several ways—for users to get answers to specific questions. What tools do users need to build queries (Boolean, relevance feedback, etc.? How can we best support the fundamentally iterative process of retrieving the desired information?

5. Alert management. How should users be alerted to time-sensitive information?
How can users create user profiles so that they system can notify them when potentially relevant information appears? Since information varies in its import, and since users will desire varying levels of 'privacy,' what sort of priority and privacy levels will provide satisfactory levels of control over interruptions?

Managing and working with information.

How do users manage, view, and work with retrieved information? Most information has some sort of structure (e.g. dictionary definitions; email conversations). Representing this structure can greatly increase the understandability and usability of the information.

6. Granularity management. How does a user get progressively more information?
Regardless of the query mechanism, a user is usually confronted with several potential answers. A user needs to be able to inspect the contents of these answers to varying levels of depth. What are the levels of information that a user needs to decide if a piece of information is useful? How should these levels be displayed/uncovered to the user? Control over granularity can also greatly facilitate browsing. (See the OED prototype [9])

7. Thread construction. How does a user follow an information "conversation?
"In e-mail and the INTERNET, it's often important to be able to grasp the development of a conversation over time. In the same way, newspapers often run follow-up stories. How should these information "threads" be depicted and made accessible to the user. How general is this problem?

8. Special purpose representations. What "meta" information is useful to users?
In the OED prototype we explored the notion of "sense graphs" where the senses of a word can be graphically displayed. We are currently discussing displaying news from the DowQuest database on a map of the world. By exploring a number of different types of databases we hope to discover whether or not there are useful special purpose representations that can be applied across the board. Alternatively, we might find that we need to design a way to consistently incorporate database-specific filters.

9. Information is more than data. What special characteristics of information should be supported?

Since one of the primary uses of information will be as cited or quoted evidence, perhaps any retrieved information ought to always carry along its citation, to facilitate inclusion in documents. If a particular set of retrieved records (or a news-story or stock quote or whatever), is likely or certain to change over time, how do users take that into account (e.g. links)?

Implementation Issues

The actual design and implementation of the portrayal is not understood at this time. It will be dependent upon both the container design and the results of the above investigations. However, we expect that it will need to be fairly closely bound with the container such that it will probably share similar development tools. Therefore, it is most likely that the portrayal will be implemented on Blue using MacApp or equivalent.

Containers

Description

The container is the central component to the architecture. Its role is to contain or hold the information pertinent to a particular question or topic. The container is bound to at least one portrayal from which it obtains its general instructions, For example, through the portrayal, the container learns about such things as topic, where to look, how much to spend, when to look, and so on. The container then worries about keeping itself filled with information that meets the criteria.

In performance of its task, the container supports 2 external protocols. The first is the portrayal to container protocol. This must be rich and general enough to allow a variety of portrayals to be "plugged in". It is bi-directional, in that the portrayal can send unsolicited messages to the container and vice versa.

The second protocol supports container to search component communication. The search agent perform queries on behalf of the container. The container passes along the criteria over this protocol to each search component. There is one for every source that the container needs to search. This protocol will initially be based upon NISO standard Z39.50. [8]

Containers are created by a portrayal when the user initiates a request for information. Once created, containers retain the state of the portrayal's request. That is, they are persistent. They are also "agents" in that they live on even if the portrayal is currently closed. Thus, they can continue to fill themselves with relevant information continuously. They can only be destroyed when the portrayal explicitly removes this particular container.

Implementation Issues

Again, we do not yet have details on the implementation of containers. In fact, recent design discussions have raised many serious questions about the loose coupling between containers and portrayals. However, it is generally recognized that containers and portrayals may need to be fairly tightly coupled for speed efficiency. Future operating systems may allow them to be loosely coupled.

As indicated earlier, close coupling with the portrayal will have significant influence on the implementation environment. At this time, the most like choice is MacApp or equivalent on a Blue platform.

Search Components/Searchers

Description

The Search Component or "Searcher" is a piece of software whose rôle in the Rosebud architecture is to connect a Container Component to a Server Component. By definition, it is local to the processor where the Container is running, and can communicate with it using a specialized protocol. The Server may be either remote or local, and the Searcher communicates with it using X39.50.

The purpose of the Searcher is to make a Server transparent to a Container so that a Container need not deal with any specialized text formats, or with remote communications.

To do this, the Searcher must:

- Take a query (text) from the Container
- Transform it to whatever form will please the Server
- Send it to the Server (possibly remote)
- Get text back (possibly in some weird format)
- Present it to the Container as local information in a standardized form, namely chunks of plain text.

(Actually the above is an oversimplification, but it gives the general idea.)

Thus the principal functional parts of a Searcher implement one-way format conversions for queries and retrieved text, and remote communications.

Searcher Variants

In order to make a Server transparent to a Container, a Searcher needs to know about certain peculiarities of the Server. For example, a particular Server might provide text in Microsoft Word format, and the Searcher must convert this to plain text.

This means that there are different kinds of Searchers. Exactly how to handle this is an implementation issue that is not yet fully understood; for example, it may be that there is really only one kind of Searcher and it can be configured on the fly to use any one of several different agents to transform the information it receives from a Server. On the other hand, this may not be a practical implementation; we need to figure out the right granularity for our agent architecture. Also note that this is a platform-dependent issue.

Implementation Issues

We assume that a Searcher will be an "agent" running initially in a Blue MultiFinder partition as a faceless background task and later as a task under the Opus 2 on the Pink platform. It is possible that it will make use of other agents implemented in the same way, e.g. there might be a single agent implementing X39.50 for multiple Searchers.

The relative simplicity of Searchers, and the likelihood of multiple instances running simultaneously, argues for implementation in a "low-level" language, i.e. C or Pascal as opposed to Smalltalk or Lisp. However, if concurrent Searchers (see below) are eventually implemented, this might be a case for using a higher-level runtime system to obtain multi-threaded execution. This is another platform dependency.

Ideally, Searchers would be created as needed (by Containers) and killed when no longer required. Presumably this will not be possible until we are running on Pink.

The most obvious implementation of Searchers is to have one Searcher running for each connection between a Container and a Server. However, it is also possible to run only one instance of each variant (see above) that is currently required by the existing connections. This would require each Searcher instance to be capable of concurrently servicing multiple Containers and/or Servers.

While the concurrent model has its attractions in terms of efficient use of resources, the simplicity of a Searcher that only knows about one Container and one Server is preferable for initial implementation.

The initial Searcher implementation will be a background application under Blue MultiFinder. Like Agent L [10], it may not be truly faceless in the first implementation, so as to include a tiny menu interface for shutting down manually and for debug control.

It will communicate with a Container (and any other local agents) via the 1990 IPC mechanism, and with a Server via the (extended) X39.50 protocol using a special transport and session.

No concurrency will be attempted initially; a Searcher will connect a single Container to a single Server.

We propose that the initial implementation be in MPW C in order to get moving quickly. Conversion to MPW C++ would be highly desirable as soon as the fundamental functionality is reasonably stable.

Server Components/Servers

<u>Description</u>

Rosebud server components are the primary sources of information. There are 3 classes of server components:

- Local servers; they make available information created and managed by your personal computer;
- Group servers; they collect and prepare information for a group of users;
- WAIS servers; Wide Area Information Servers; they offer the huge amount of commercially available information.

Server components execute independently from all other components. Once created they run "forever" until explicitly destroyed. They execute continuously. That is, they are constantly monitoring their sources for the arrival of new information to be processed. They are also always available for search requests.

Servers collect information in various "raw" forms such as text, numerical data, or visual information. It is the server's responsibility to understand the raw information and massage it in specialized ways so as to make it available for future searching. It is expected that there will be many classes of servers to accommodate the diverse forms or "raw" data.

Once the "raw" data has been processed by a server, it can be offered to the client community. The clients are *Search Components*. Search components are more fully described elsewhere. Server components offer their information to search components using a standard server protocol. This protocol will initially be based upon the NISO standard Z39.50. The server component will implement the "target" side of Z39.50.

Server components can be accessed over arbitrary communications networks. These networks may be high performance local networks or more loosely coupled systems using modems or other remote communications mechanisms. It is the servers' responsibility to encapsulate and manage the particulars of its communications facilities. It is expected that this would be accomplished through the use of specialized internal components.

Local Server Functions

Every personal computer has one (or more) personal document server component(s). These local servers process the contents of all of the documents "owned" by this particular computer. The contents may be text, numerical data, image or graphical. There could be a special class of server for each type of document. For example, one server may specialize in processing spreadsheet data while another may process text. Alternatively, there may be only one large local server which has subcomponents for each data type.

For the purposes of the Rosebud prototype, there will be more than 1 local server. This will force us to discover issues that arise with multiple local servers. We expect multiple local servers in the future as third parties construct special local document processing components to match with their new document types.

The local servers must be capable of at least "reading" the contents of their respective sources. The server must be able to process the text inside a MacWrite, MS-Word, Full Write etc. document. The ability to process information beyond the text, such as fonts, will be optionally done for Rosebud. The local server will also attempt to "read" the text inside Excel spreadsheets and MacDraw pictures. There will be no attempt initially to process spreadsheet data at this stage.

Group Server Functions

The group server processes information from multiple sources on the behalf of multiple clients. The server manages the low-level communications to the raw data source. For example, it contains subcomponents to deal with modems and/or the IBX for the purpose of retrieving AppleLink mail.

As with local servers, it is expected that there will a group server class for each raw information source. The group server will encapsulate specialized functions needed to interpret its particular source. In Rosebud, there will be a group server for AppleLink, Usenet bulletin boards and OED/Groliers reference data.

Although referred to as group servers, it is expected that they would be implemented in almost exactly the same way as local servers. The reason they are group servers is that a community can share disk storage and processing power across a number of common sources, freeing personal computer horsepower for local server processing.

Group servers can also become the central processing point for documents published by the community to be shared. The same document processing subcomponents deployed by local servers would be used here. By processing these shared documents on the group server, they need only be analyzed once to enable searching for the whole community.

Group servers will be implemented on 2 platforms. The Macintosh will serve as a platform in most cases. However, in conjunction with Thinking Machines, a Connection Machine offering the same group server protocol will be available for some aspects of the project.

WAIS Server Functions

It is expected that the most common source for information will be from Wide Area Information Servers (WAIS). Currently, WAIS represent many Gegabytes of text information. It is typically accessed through iteration with a query engine provided at the WAIS site. The raw information is rarely made directly available.

We would prefer to see a WAIS in exactly the same way as we see a group server. It would, of course, represent a much larger information source, but could be searched in the uniform way. This will require cooperation from the WAIS.

As a joint project with Thinking Machines, Rosebud will connect to a prototype WAIS providing information published by Dow Jones. A Connection Machine will provide the query processing at Dow Jones. Software will be written to offer the Z39.50 target protocol.

Implementation Issues

The following components need to be implemented:

- Local server for MacWrite, MS-Word;
- Local server for Excel and MacDraw;
- Group server for AppleLink;
- Group server for Usenet;
- Group server for OED/Groliers.

Many of these components will want to share subcomponents. Examples are:

—PLS/IR engine for text indexing;
—TCP/IP communications
—IBX/modem communications

This next phase will be prototyped on Blue System 6.0. For space and speed reasons, and because these servers are not overly complex, we plan to implement them in Blue C++. This should make it easier to interface with critical support software like the communications toolbox and PLS. Each component will be written as a MultiFinder background task and intercomponent communication will be via IPC 1990.

Support Utilities

We have described the four major components at the heart of the Rosebud architecture. In support of these components, we will need to provide a suite of utilities. There are roughly 5 areas in which we will provide support:

- Persistent data
- Communications
- Information retrieval
- Searcher/Server application protocol (Z39.50)
- Document specific text translators

Persistent Data

The contents of containers clearly need to be kept around as long as the container is in existence. If a container is copied or moved onto a floppy, its contents need to be carried with it. The portrayal will make references to the contents of containers. Clearly, we need a mechanism which ensures that containers can made permanent and whose contents can be shared across components. We plan to make use of the Leopard persistent object model to help us with this process.

Communications

We have to be able to connect components together within a processor, across the LAN, and across the country.

For local and LAN communications, we will employ IPC 1990 and the base level mechanism. On top of IPC 1990, we will implement a transport and session protocol to allow complete application to application communication.

Dow Jones offers 2 distinct information services. The first, Dow Vision, is a news feed service. It is a broadcast service in which information is transmitted in real-time to the destinations. The second service is Dow Quest. In this service, we, in effect, talk to a Connection Machine which acts as the server component.

In both cases, we connect with Dow Jones via X.25. We will use the Apple X.25 communications package in conjunction with the communications toolbox. For DowVision, we will implement the Dow Jones DowVision Broadcast Specification [11]. For DowQuest, we will implement a different transport and session protocol to be specified by Dow Jones.

To access local information sources, such as Usenet, we will implement TCP/IP connections between server components and our VAX. We plan to model this code after the already existing mechanisms used by HyperNetNews.

Information retrieval

A key technology used to analyze and retrieve based upon content is the information retrieval engine. The earlier Hearst Phase II prototype was based upon a commercial product called Personal Librarian Software (PLS) [12]. As a result of that usage, we decided to acquire the source for the product so that it could be adapted to our particular needs. This process is now underway. We, therefore, plan to continue to use PLS in the Rosebud prototype. A small library to translate server components IR engine calls into the particular PLS calls is planned.

Brewster Kahle of Thinking Machines, has produced a simple IR engine for his own use in testing. We may consider using this IR engine for the WAIS demonstration since the availability of PLS source remains a risk item. If we cannot obtain a PLS version which meets our needs, we have a fall-back plan to adapt Brewster's engine to the prototype.

Searcher/Server application protocol (Z39.50)

The high level application protocol between searchers and server components is to be based upon NISP Z39.50 [8]. Thinking Machines will produce a protocol specification adapted from Z39.50 to meet our needs. Thinking Machines programmers will implement the server end for the Connection Machine. Apple will need to implement the client side for use by the searchers as well as the server side for local and group server components.

Document specific text translators

Finally, if the Rosebud prototype is to help with managing local files by content, then local documents will need to be processed by a local file server component. This server will require help translating specific application file formats. For example, Microsoft Word stores its text differently from MacWrite. For each application type that we might want to process, we will need to write a translator.

WAIS

A Rosebud framework will enable the creation of a new suite of applications built around the ideas of content analysis. While Rosebud applications would be useful managing the information created and kept locally by a user, its true value emerges when those same applications can help organize the sea of publicly available information. Therefore, a prime requirement for the Rosebud prototype is the ability to integrate with powerful non-Apple servers and large informations publishers.

Thinking Machines is a manufacturer and supplier of high performance information servers. Apple Computer is a manufacturer and supplier of powerful, easy to use personal computers. Together, the two companies can explore and develop systems which will bring the value of the information age to the average person.

Joining us in this task is Dow Jones and Peat Marwick. Dow Jones is a publisher of a large amount of electronic information including *The Wall Street Journal.* Peat Marwick is a large Macintosh user community with a great need for managing vast amounts of information. The four companies will cooperate to test a Rosebud prototype in real users hands.

Apple and Thinking Machines intend that their joint efforts will result in specific protocols, products, services and applications for Wide Area Information Servers (WAIS). In particular, we seek to:

- Publicly demonstrate the use of the Macintosh personal computer to access, filter and organize information offered from powerful information servers;
- Develop prototype, standard protocols between Apple clients and Thinking Machines servers;
- Develop an example user interface which demonstrates access of both local and remote information through a single user model; and
- Test the principle of distributing information processing control across agents on different vendors' equipment.

The user interface to be implemented as part of the WAIS portion of Rosebud is not yet defined. It is expected to be inspired by the work done for the portrayal component. While we expect to use many of the Rosebud components for WAIS, we will likely spin off a special portrayal to meet the WAIS schedule and any special needs of Peat Marwick.

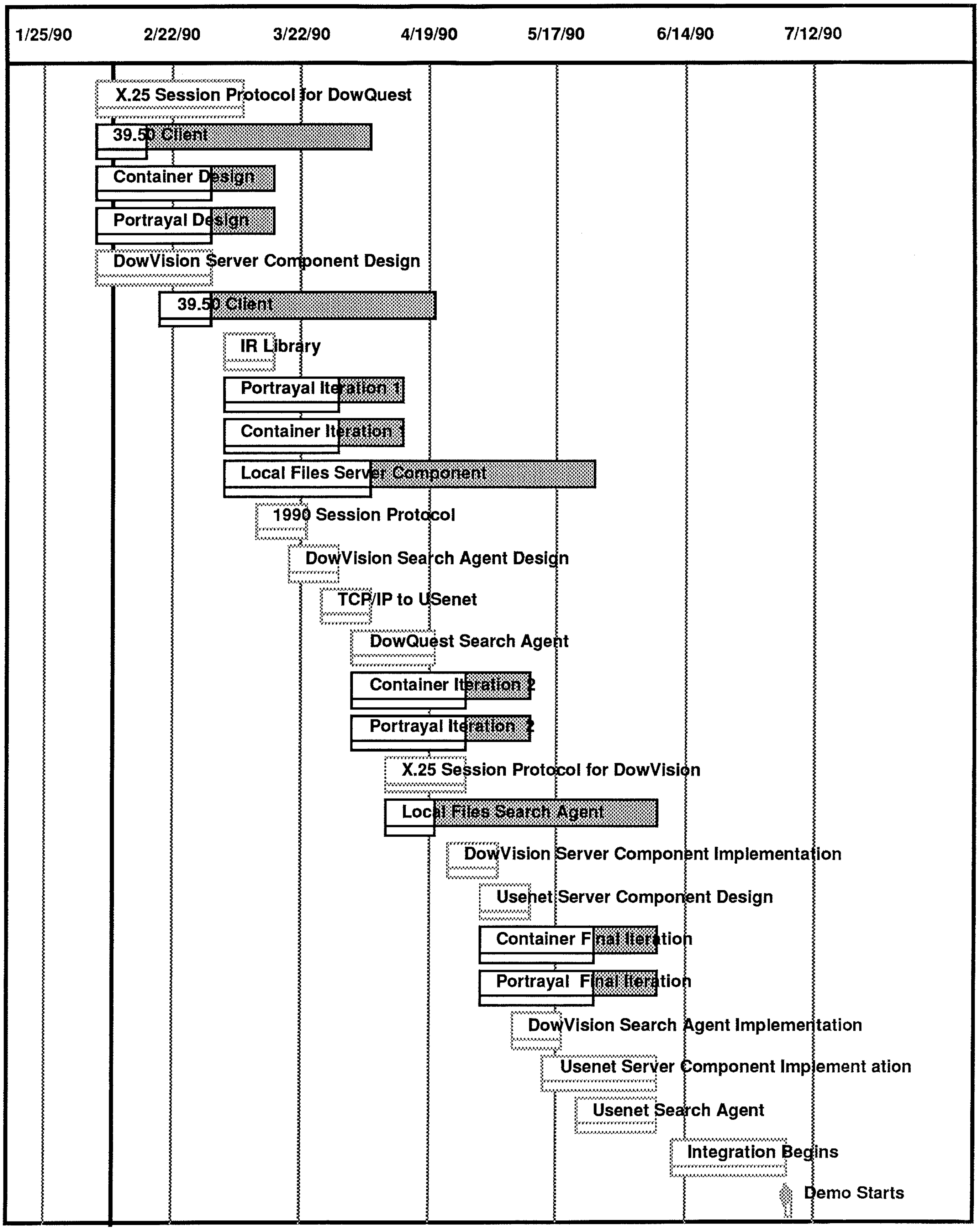
Schedule

In general, the Rosebud schedule is divided into major phases of about 6 months apiece. Presuming the Hearst/Red efforts as history, we are now entering Phase III. This is expected to be complete in about 4-6 months. Phase III will focus upon extending the prototype to connect to "live" information sources including non-Apple sources. The technical details for this Phase of the prototype are more fully described in the earlier sections. We have included here the detailed overall schedule for Phase III as well as the individuals' schedules.

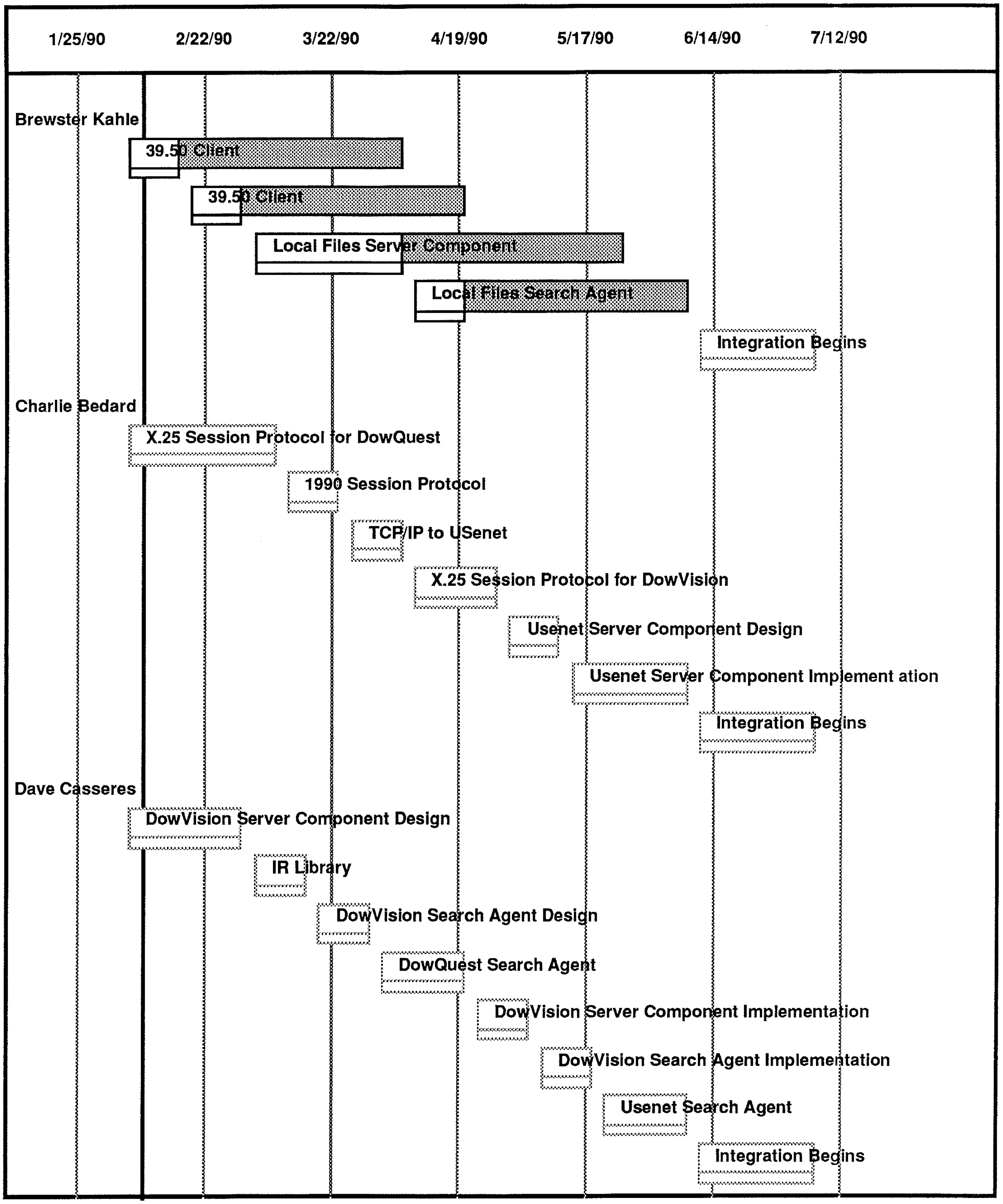
Phase IV will be the first attempt for Rosebud to execute upon a Pink platform. While we don't expect to move all of Rosebud to Pink in 1 step, we hope that we will be able to move certain key components to Pink. For example, servers components and search components exhibit a large amount of concurrency in their behavior. We would certainly prefer to take advantage of Pink facilities in this area. Also, we would like to take PLS, which we will have the source to, and update it to deal with persistent objects rather than specialized files. That way, we will find it much easier to reference items in containers and portrayals.

Phase V will begin approximately one year from now. It will consist of moving the portrayal and any other remaining prototype components from Blue to Pink. At that point, we expect that the Ralph programming language will be available for Pink. We will use Ralph to create the Pink based user interface(s).

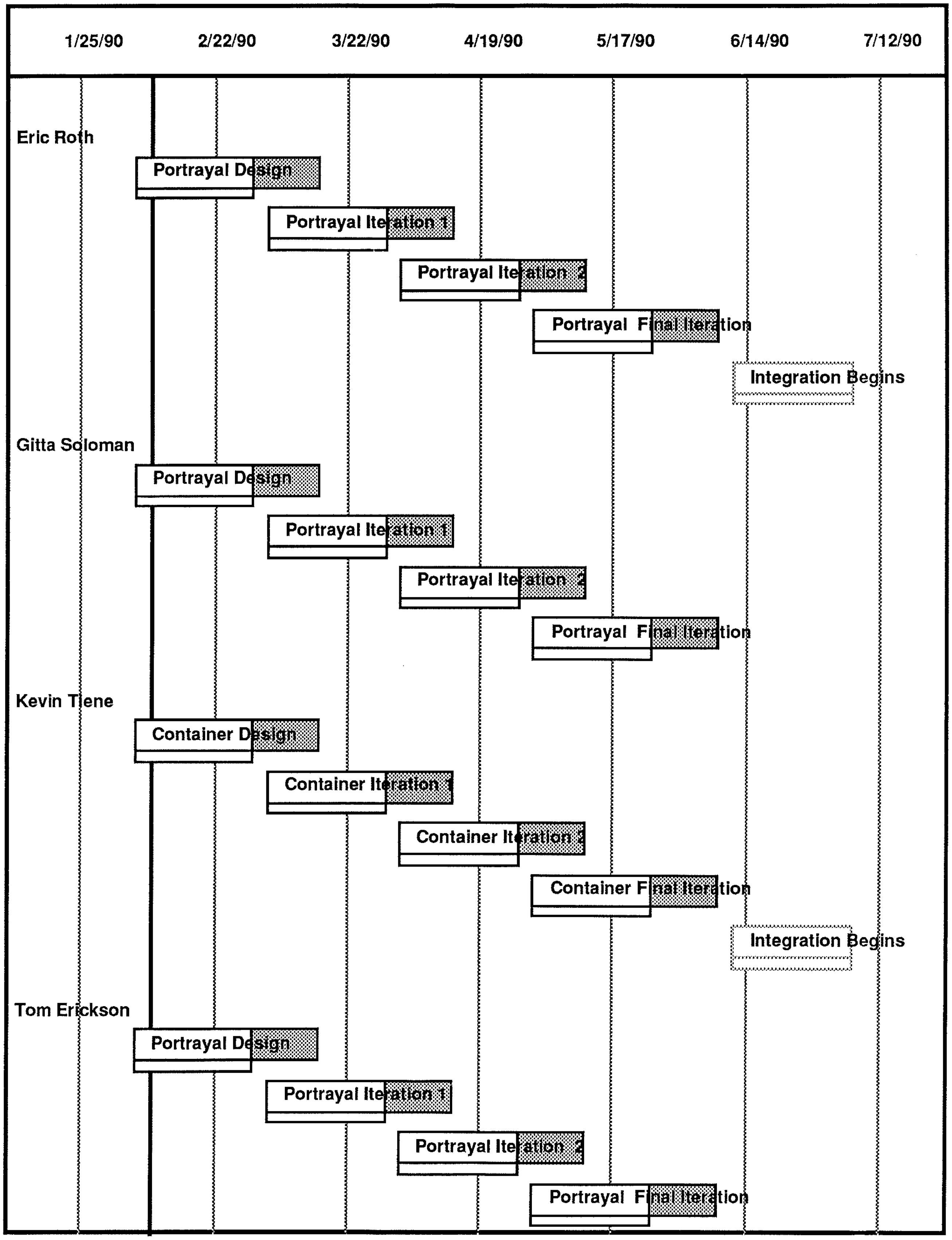
Although not explicitly noted, we fully expect that the architecture and design of the various components will change with each iteration of the prototype. We also assume that each Phase will be successful and will warrant progression to the next Phase. At the completion of Phase V we feel that we will have a powerful and compelling prototype executing on a Pink system and capable of running on our most powerful CPU platforms. At that point, we should consider moving Rosebud or some portion of its technology into Product Development.



Overall Schedule



Individual Schedules (1)



Individual Schedules (2)

Risks and Issues

As with any project, there are some risks. Some of these issues have immediate impact on Rosebud. Others will have impact further into the project. In all cases, however, their outcome will affect both the functionality and the project schedule.

Our most immediate concern is with the development tools that we will be using for much of Phase III. Most of the implementation with be in MPW C , C++, and/or MacApp on Blue platforms. Few programmers in the team are familiar with all of the tools. There will a significant learning curve that will take precious time from the schedule. Also, in anticipation of a future port to Pink, we will significant programming in C++. This represents a significant learning curve for all of us since no one has used C++ before.

A key technology in Rosebud is the Information Retrieval engine. our current plans call for continuing use of PLS [12]. While we can make some progress using the object code that we have today, we know that we will need to get inside the engine to make it truly useable for the demos and our own day-to-day use. Therefore, we must receive access to new object code before May. Since the PLS contract is not yet signed (although imminent), we may have to move to our backup IR strategy later in the project.

We expect that we can squeak through Phase III with limited support for persistent objects. We would prefer, however, to use them wherever possible so as to gain experience with Leopard. Subsequent phases of Rosebud will depend much more on persistent objects. We hope to be able to update PLS to use Leopard objects for its documents in Phase IV. There remains a big question as to whether Leopard will be properly supported. We now have Apple effort devoted to Leopard, but we will still require an ongoing association with Instantiations. The current contract, while moving ahead at a maintenance level, is insufficient for us to complete some missing remaining functionality.

Not an issue for Phase III, but certainly a year from now is the issue of Ralph. We currently expect to be able to move to Pink by the late summer, early fall of 1990. In 1991, however, we want to be able to move to Ralph on Pink. At this stage, Ralph still looks shaky and there does not yet exist a plan for running Ralph on Pink. The Rosebud team has contributed 2 members, Dave Smith, and Joshua Susser, to help with the Ralph effort. This signifies our strong support for the technology and our commitment to ensure that it will happen.

Finally, there is the issue of human resources. The current project team is made up of team members from various functional groups. While each team member is certainly enthusiastic about Rosebud, not everyone is doing Rosebud exclusively. This makes scheduling even more inaccurate as it is hard to estimate when less than 100% time is available. Also, there is a concern that changes in functional group priorities cab affect a team members potential participation. We're hoping, however, that a continuing strong commitment to Concept Systems will remain and that Rosebud will maintain a reasonably high priority.

Bibliography

[1] *Overview of RED* — Dave Smith and the Red Project Team; March 8, 1989

[2] *The Information Access Concept System* — Study group Report; April 27, 1989

[3] *The Dynamic Library* — Dave Smith and the Red Project Team; April 19, 1989

[4] *What is Red?*— Charlie Bedard; July 11, 1988

[5] *Red Scenario* — Dave Smith; March 30, 1989

[6] *Red Presentation* — Dave Smith and the Red Project Team; March 13, 1989

[7] *Catalyzing a Market of Wide Area Information Servers (WAIS)* — Brewster Kahle; August 7, 1989

[8] *Information Retrieval Service Definition and Protocol Specifications for Library Applications* — American National Standard Z39.50; National Information Standards Organization (NISO); Approved January 15, 1988

[9] *The Oxford English Dictionary Prototype* — Gitta Soloman and Tom Erickson; Video Tape; August, 1989

[10] *Hearst Phase II Prototype*— The Hearst Project Team; September, 1989

[11] *DowVision Broadcast Specification*— Dow Jones News Retrieval Service; August 14, 1989

[12] *CPL: Callable Personal Librarian Application Programmer's Interface Version 2.0* — Personal Library Software Inc.; January, 1989